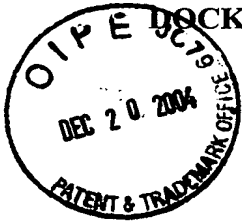


AF  
JCS



DOCKET NO.: MSFT-0510/36349.1

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

**In Re Application of:**

Joseph P. Fernando et al.

**Application No.:** 09/240,406

**Filing Date:** January 29, 1999

**For:** Extensible Object Model

**Confirmation No.:** 7291

**Group Art Unit:** 2126

**Examiner:** Sue X. Lao

DATE OF DEPOSIT: December 17, 2004

I HEREBY CERTIFY THAT THIS PAPER IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE AS FIRST CLASS MAIL, POSTAGE PREPAID, ON THE DATE INDICATED ABOVE AND IS ADDRESSED TO THE COMMISSIONER FOR PATENTS, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450.

TYPED NAME: Vincent J. Roccia  
REGISTRATION NO.: 43,887

MS Appeal Brief - Patent  
Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

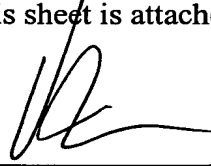
**APPLICANTS REQUEST TO TRANSFER APPEAL BRIEF  
ERRONEOUSLY FILED IN USSN: 09/240,046 TO USSN: 09/240,406**

On December 16, 2004 Applicants filed an Appeal Brief for the above-identified matter. However, after the documents were mailed it became apparent that the Serial No. was incorrect. Applicants hereby request to transfer the APPEAL BRIEF erroneously filed with The United States Patent and Trademark Office in Application Serial No. 09/240,046 to Application Serial No. 09/240,406. Applicants respectfully requests the Commissioner to take whatever steps are necessary to rectify this error. Enclosed herewith are copies in triplicate of the documents as filed on December 16, 2004.

**DOCKET NO.: MSFT-0510/36349.1**

**PATENT**

Should any fee be deemed necessary to grant this Request, the Commission is hereby authorized to charge payment of the above fees associated with this communication or credit any overpayment to Deposit Account No. 23-3050. This sheet is attached in triplicate.



Date: December 17, 2004

---

Vincent J. Roccia  
Registration No. 43,887

Woodcock Washburn LLP  
One Liberty Place - 46th Floor  
Philadelphia PA 19103  
Telephone: (215) 568-3100  
Facsimile: (215) 568-3439

© 2004 WW



DECKET NO.: MSFT-0510/36349.1

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: **Joseph P. Fernando**  
et al.

Confirmation No.: 7291

Serial No: 09/240,046

Group Art Unit: 2126

Filing Date: January 29, 1999

Examiner: Sue X. Lao

For: Extensible Object Model

COPY

DATE OF DEPOSIT: December 16, 2004

I HEREBY CERTIFY THAT THIS PAPER IS BEING  
DEPOSITED WITH THE UNITED STATES POSTAL  
SERVICE AS FIRST CLASS MAIL, POSTAGE PREPAID,  
ON THE DATE INDICATED ABOVE AND IS  
ADDRESSED TO THE COMMISSIONER FOR PATENTS,  
P.O. BOX 1450, ALEXANDRIA, VA 22313-1450.

TYPED NAME: Vincent J. Roccia  
REGISTRATION NO.: 43,887

Mail Stop Appeal-Brief Patents  
Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

**APPELLANT'S BRIEF PURSUANT TO 37 C.F.R. § 1.192**

This brief is being filed in support of Appellant's appeal from a final rejection of claims 16-21 and 27-44, dated March 23, 2004. A Notice of Appeal was filed on July 16, 2004.

This appeal brief is being submitted in triplicate, pursuant to 37 C.F.R § 1.192(a). Appellant respectfully requests that the final rejection be reversed and that the application be remanded to the examining group for allowance.

**I. REAL PARTY IN INTEREST**

The real party in interest in the present appeal is Microsoft Corporation by virtue of an assignment from applicants Joseph P. Fernando and Chris Fraley to Microsoft

Corporation, filed on March 25, 1999 and recorded on April 1, 1999 at Reel 9869/ Frame 0768.

## II. RELATED APPEALS AND INTERFERENCES

There are no other appeals or interferences known to the Appellant, the Appellant's legal representative, or the Assignee that will directly affect or be directly affected by or have a bearing on the Board's decision in the present appeal.

## III. STATUS OF CLAIMS

- A. Claims 16-21 and 27-44 are pending. Claims 16-21 and 27-44 are reproduced in Appendix A attached hereto. There are six independent claims: 16, 27, 29, 34, 42 and 44.
- B. No claims stand allowable.
- C. Claims 16, 18-20, 27, 28, 42 and 43 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Pat. No. 6,289,500 to Baxter *et al.* ("Baxter") in view of U.S. Pat. No. 5,761,499 to Sonderegger ("Sonderegger").
- D. Claim 34 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Pat. No. 6,275,979 to Graser *et al.* ("Graser") in view of Sonderegger.
- E. Claims 17, 29-33, 35-41 and 44 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over the "IBM San Francisco Framework" as disclosed by Baxter and Graser in view of Sonderegger.
- F. Claim 21 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over Baxter in view of Sonderegger as applied to claim 16 and further in view of Schmidt *et al.* ("An Object-Oriented Framework for Developing Network Server Daemons") ("Schmidt").
- G. Claims 16-21 and 27-44 are the subject of the appeal.

## IV. STATUS OF AMENDMENTS

No amendments have been filed subsequent to the office action dated March 23, 2004.

**V. SUMMARY OF INVENTION**

Object-oriented programming languages are pervasive in today's computing environment. As is well known by those skilled in the art, an object is a module of code that provides a definition or a template for a type of data to be stored. The object is externally accessible through well-defined and controlled connection points, known as the object's methods and properties. Various software application development environments expose object models whose limited properties and/or methods are fixed or "inherent" in that they are not extensible by external software packages.

The present invention overcomes such a limitation by extending functionality to an object in an object model. In particular, the present invention provides an extension mechanism that can be bound to a particular object in an application object model to provide functions beyond those that are standard in an application development environment. For example, such additional methods and/or properties may be directed to objects that form the application itself. Also, the extension mechanism of the present invention allows the extension's code to be dynamically added during a run-time environment.

As recited in claim 16, for example, the present invention provides a system for extending the functionality of a class object. The functionality of the class object is extended using, *inter alia*, an extensible object model that creates an extension object from an existing extension package when a requested functionality is not inherent in the class object. The extension object then extends the class object to provide the requested functionality.

A benefit provided by just one example embodiment of the present invention is allowing extension objects from one vendor's application to be available to another vendor's application. By extending the functionality of a class object and creating an extension object from an extension package when a requested functionality is not inherent in

the class object, this particular embodiment allows for extending methods and/or properties of an object residing at any level in an application object through an extension object.

For example, in one embodiment described in the present specification, a hierarchical object model is extended. The hierarchical object model may include a window object with inherent methods, like “close” window, “move” window,” and “resize” window.

(*Specification - page 9, lines 10-14*). The inherent methods of a window object 203 may be extended by an “editor” method via an editor extension object 204. (*Specification - Figure 2 and page 9, lines 15-23*). The editor extension object 204 locates an editor package 206 that is registered on an extension database 205, and obtains an extension object from the editor package. (*Specification - page 9, lines 15-23*). When the window object 203 is first invoked by an application, it is determined that a certain editor method is not inherent in the window object 203. (*Specification - page 10, lines 1-5*). The run-time environment then searches the extension database 205 to locate the editor package 206. (*Id.*). The editor package 206 signals the extension provider object 208 to create the editor extension object 204, so as to service all calls to the editor method. (*Id.*).

## VI. ISSUES

- A. Whether claims 16, 18-20, 27, 28, 42 and 43 are patentable over Baxter in view of Sonderegger.
- D. Whether claim 34 is patentable over Graser in view of Sonderegger.
- E. Whether claims 17, 29-33, 35-41 and 44 are patentable over the “IBM San Francisco Framework” as disclosed by Baxter and Graser in view of Sonderegger.
- F. Whether claim 21 is patentable over Baxter in view of Sonderegger as applied to claim 16 and further in view of Schmidt.

**VII. GROUPING OF CLAIMS**

**Group 1:** All of the pending claims stand or fall together. Claims 16-21, 27-28 and 42-43 include the feature of determining whether a requested functionality is inherent in the class object and creating an extension object from an existing extension package, when a requested functionality is not inherent in the class object. Claims 29-41 and 44 use similar language and include the feature of determining whether a requested functionality is inherent in the class object and directing the request to the functionality in a second extension object, when the functionality is not available in the first extension object.

**VIII. ARGUMENT**

**A. The 35 U.S.C. § 103 Rejections of Group 1 (Claims 16-21 and 27-44)**

Claims 16, 18-20, 27, 28, 42 and 43 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Baxter in view of Sonderegger. In particular, the office action suggests, *inter alia*, that while Baxter does not teach an extensible object model that determines whether a requested functionality is inherent in a class object, Sonderegger does teach such an extensible object model. (*Office Action dated March 23, 2004* at p. 2). In particular, the office action suggests that column 8 lines 45-61, column 9 lines 12-32 and column 10 lines 31-38 of Sonderegger teach “determining whether a requested functionality is inherent (registered in the registry file 44/OLE registry file) in the class object (check the registry file 44 for availability of the desired COM component).” (*Office Action dated March 23, 2004* at p. 2-3). To summarize, the office action makes the following three contentions with regard to Sonderegger, citing column 10, line 39 to column 11, line 10:

- the claimed “[i]f the requested functionality is not inherent in the class object” is taught by Sonderegger’s “the desired COM component not registered in registry file 44”;

- the claimed “an extension package is located” is taught by Sonderegger’s “query directory services and database 52 for unregistered COM/OLE components on component server”; and

- the claimed “creat[ing] an extension object” is taught by Sonderegger’s “COM interface IClassFactory.”

(*Office Action dated March 23, 2004 at p. 2-3*).

However, with all due respect, each of these contentions are unsupportable and simply incorrect.

It should also be noted that the Examiner further suggests that one of ordinary skill in the art would recognize that “any functionality inherent to an object is the functionality which is offered by the object and is available for invocation.” (*Office Action dated March 23, 2004 at p. 7*).

Applicant does not argue this general point regarding the art. However, the Examiner goes on and further supports the three contentions above by suggesting that Sonderegger teaches that the “functionality/services registered in the registry file 44 are the functionality inherent to the component/object.” (*Office Action dated March 23, 2004 at p. 7*). With all due respect to the contentions in the office action, applicants respectfully disagree.

A close reading of the entirety of Sonderegger and the relevant portions cited in the office action in support of the Examiner’s contention reveals, quite simply, these concepts are wholly distinct from the present invention.

As its very title indicates, the context in which Sonderegger operates is globally distributed software. In particular, Sonderegger contemplates a method for managing globally distributed “software components in a network of computers such as . . . the Internet.” (*Sonderegger – Abstract*). In other words, Sonderegger is concerned with the locating of software components, like software objects, across large networks like the Internet.

Sonderegger explains that each software component includes a binary object and at least one interface to the binary object, and that the network includes “a registry file identifying currently registered components and their locations in the network.” (*Sonderegger – Abstract*). The network also includes a component server computer on which unregistered components reside.



Sonderegger explains that unregistered components may be located on the network by conducting “a search based on the class identifier of a desired component.” (*Sonderegger* – Abstract). Once an unregistered component is located by the search and selected, it is registered and made available to clients. (*Sonderegger* – Abstract).

Apparently, because Sonderegger operates over such a vast network like the Internet, a separate file called the “registry file” identifies the location of registered software components. (*Sonderegger* – column 3, lines 17-19). Sonderegger explains that “OLE software components are accessible on a given computer only if they are registered in a ‘registry file’ which has a very specific format.” (*Sonderegger* – column 2, lines 5-7). Sonderegger’s “registry file maps software component identifiers known as ‘CLSIDs’ to the file system location of the identified server’s binary object and to ‘type library’ information about that server component’s interface.” (*Sonderegger* – column 2, lines 14-17). It is well known to those skilled in the art, that CLSIDs are Internet-specific identifiers that provide a globally unique identifier used in a unique resource locator (URL) scheme to identify Component Object Model (COM) class objects in servers. Sonderegger’s “registry file” provides a mapping of these CLSID identifiers to a location on the Internet, for example. This background of Sonderegger provides proper context and further explains why the sections of Sonderegger quoted in the office action do not teach the present invention.

First, Sonderegger’s teaching of determining whether functionality is “*registered* in the registry” is not commensurate with the present invention’s determination of “whether a requested functionality is *inherent* in a class object.” Specifically, with all due respect to the contentions in the office action, column 8, lines 37-56 of Sonderegger does not teach or even suggest whether a requested functionality is “inherent” in a class object for the purposes of extensibility. (*Office Action dated March 23, 2004* at p. 2) (“whether a requested functionality

is inherent (registered in registry file 44/OLE registry file) in the class object (check the registry file 44 for availability of desired COM component”). Instead, Sonderegger’s discussion at column 8, lines 37-56 with regard to the “registry file” refers to whether a client can access a particular component on a network, like the Internet. (*Sonderegger* – column 8, lines 49-56) (“Thus, any of the components 48 which do not appear in the registry file 44 are not available to the COM client 40, even if the client 40 has access through the network to the files that contain the components 48.”). Therefore, contrary to the contention in the office action, Sonderegger’s teaching of determining whether functionality is *registered* in the registry is not commensurate with the present invention’s determination of whether functionality is *inherent* in a class object. As acknowledged in the office action, the same is true of Baxter. In fact, the word “inherent” does not even appear in either Sonderegger or Baxter.

Second, and contrary to the suggestion in the office action, Sonderegger’s discussion at column 10, lines 39 to column 11, line 10 does not suggest that if certain functionality is not inherent in the class object, an extension package is located and creates an extension object that provides the requested functionality. Quite the contrary, even if one disagrees with applicants’ first argument, and somehow assumes that Sonderegger’s “registered in the registry” is commensurate with “determining whether requested functionality is inherent in a class object,” Sonderegger still cannot be said to locate an extension package to create an extension object that provides the desired functionality. Instead, Sonderegger admits that “[i]f no database entry or object 54 corresponding to the specified COM component identifier is located by the search . . . [t]he client 40 must then proceed as best it can without the desired services.” (*Sonderegger* - column 11, lines 6-10).

Moreover, the search of the database that Sonderegger is referring to is not even close to being commensurate with the present invention’s locating of an extension package to

create an extension object. Instead, Sonderegger is referring to a search conducted on a network, like the Internet, by a client-user. If the user cannot find a component on the network, it will "return an error message" and "initiate steps 114-120," which include querying a database to locate database objects with a desired component's CLSID and to update the registry file that is local to the client. (*Sonderegger* - column 10, lines 38-42 and lines 30-37). This database search may be accomplished by a number of familiar methods including "*Internet search engines* such as Gopher, Archie, WAIS, or a World Wide Web browser." (*Sonderegger* - column 11, lines 2-5) (emphasis added). This simply is not the same feature contemplated by the present invention.

As Sonderegger further explains, this searching process may be done automatically when "the function CoGetClassObject() is altered to search the directory services and/or Internet database 52 if the desired component is not registered. If the search is successful, the altered CoGetClassObject() transfers the located component as needed, updates the registry file 44, and then passes control to the original CoGetClassObject() code." (*Sonderegger* - column 10, lines 51-57). Therefore, when Sonderegger refers to "extending the registry," it means "extend[ing] the reach of the registry file 44 beyond a single computer or file system to a surrounding LAN, NetWare Connect Services network, and/or the Internet." (*Sonderegger* - column 10, lines 59-65). Again, this simply is not the same process contemplated by the present invention.

Finally, the office action suggests that the "COM interface IClassFactory" is the same as the extension object of the present invention. Sonderegger refers to IClassFactory as one of the COM interface management services. (*Sonderegger* - column 9, lines 27-31). As is well known to those skilled in the art, the IClassFactory is a very specific type of standard COM interface that contains two methods (*i.e.*, CreateInstance and LockServer) that interact

with an entire class of objects, which are identified by a CLSID. Neither of these two methods, or even IClassFactory generally, is remotely related to the notion of creating an extension object, as is suggested in the office action.

Accordingly, applicants respectfully request withdrawal of the rejection of claims 16, 18-20, 27, 28, 42 and 43 under 35 U.S.C. 103(a) over Baxter in view of Sonderegger.

Next, independent claim 34 stands rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 6,275,979 to Graser *et al.* ("Graser") in view of Sonderegger. More specifically, the office action alleges that support for the rejection of claim 34 is found in the "discussion of claim 16 for the determining step and a motivation to combine the teachings of Graser / IBM San Francisco framework with Sonderegger." (*Office Action dated March 23, 2004* at p. 4). Accordingly, for the same reasons discussed above with respect to the rejection of claims 16, 18-20, 27, 28, 42 and 43 under 35 U.S.C. 103(a) over Baxter in view of Sonderegger, applicants respectfully request withdrawal of the rejection of claim 34 under 35 U.S.C. 103(a) over Graser in view of Sonderegger.

Next, claims 17, 29-33, 35-41 and 44 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Baxter and Graser in view of Sonderegger. Claim 17 is dependent upon independent claim 16. Claims 30-34 are dependent upon independent claim 29. Claims 35-41 are dependent upon independent claim 34. Claim 44 is an independent claim.

The Examiner cites the same reasons for rejection over Sonderegger as discussed above with regard to claims 16, 18-20, 27, 28, 42 and 43 under 35 U.S.C. 103(a). The Examiner further suggests that "both Baxter and Graser describe the run-time operations of the same IBM San Francisco framework." (*Office Action dated March 23, 2004* at p. 4).

For the same reasons discussed above with respect to the rejection of claims 16, 18-20, 27, 28, 42 and 43 under 35 U.S.C. 103(a) over Baxter in view of Sonderegger, applicants

respectfully request withdrawal of the rejection of claims 17, 29-33, 35-41 and 44 under 35 U.S.C. 103(a) over Baxter and Graser in view of Sonderegger.

Also, claim 21, which is dependent upon independent claim 16 stands rejected under 35 U.S.C. 103(a) as being unpatentable over Baxter in view of Sonderegger “as applied to claim 16” and further in view of Schmidt. (*Office Action dated March 23, 2004* at p. 6). For the same reasons discussed above with respect to the rejection of claims 16, 18-20, 27, 28, 42 and 43 under 35 U.S.C. 103(a) over Baxter in view of Sonderegger, applicants respectfully request withdrawal of the rejection of claim 21 under 35 U.S.C. 103(a) as being unpatentable over Baxter in view of Sonderegger and Schmidt.

**IX. APPENDIX**

1-15. (Canceled)

16. (Presently Amended) A system for extending functionality of a class object, comprising:

a processing unit;

a system memory in communication with the processing unit via a system bus;

a computer-readable medium in communication with the processing unit via the system bus; and

an extensible object model executed from the computer-readable medium by the processing unit, wherein the extensible object model determines whether a requested functionality is inherent in the class object, and wherein the extensible object model causes the processing unit to create an extension object from an existing extension package when a requested functionality is not inherent in the class object, and wherein the extension object extends the class object to provide the requested functionality.

17. (Previously presented) The computerized system of claim 16, wherein the extensible object model further causes the processing unit to notify the extension object when the extension is deleted.

18. (Original) The computerized system of claim 16, wherein the extensible object model further causes the processing unit to register the extension package in an extension database stored on the computer-readable medium.

19. (Original) The computerized system of claim 16, wherein the extensible object model further causes the processing unit to store the extension object in system memory when the corresponding extension is first referenced.

20. (Original) The computerized system of claim 16, wherein the extensible object model further causes the processing unit to create the extension object from the extension package by causing the processing unit to create an extension provider object and causes the processing unit to create the extension from the extension provider object.

21. (Original) The computerized system of claim 16, wherein the extensible object model further causes the processing unit to create an event filtering and sourcing object to handle events generated by the extension object.

22-26. (Canceled)

27. (Presently Amended) A computer-readable medium having stored thereon computer-executable components comprising:

an extensible object;

an extension database having an entry for an extension for the extensible object; and

an existing extension package having an interface for obtaining an extension object that provides the extension for the extensible object, wherein the extensible package determines whether a requested functionality is inherent in the class object.

28. (Original) The computer-readable medium of claim 27, further comprising:

an extension provider object that proffers the extension object as a result of a call to the interface in the extension package.

29. (Presently Amended) A method for extending functionality of a class object in a run-time environment, comprising:

determining whether a requested functionality is inherent in the class object;  
receiving a request from an application for functionality that is not inherent in the class object;

determining if the functionality is available in a first extension object;

obtaining an existing extension package having computer-executable instructions associated with the extension object functionality, wherein the extension package proffers an extension provider object when the functionality is requested;

specifying parameters to the extension provider object to create a second extension object; and

directing the request to the second extension object.

30. (Previously presented) The method of claim 29, further comprising registering the extension package in an extension database.
31. (Previously presented) The method of claim 29, further comprising storing the extension package in an extension database.
32. (Previously presented) The method of claim 31, further comprising searching for an entry associated with the functionality in the extension database to determine if the functionality is available in the extension object.
33. (Previously presented) The method of claim 29, further comprising creating the second extension object when the extended functionality is first referenced, and locating the second extension object when the extended functionality is subsequently referenced.
34. (Previously presented) A method for extending functionality of a class object in a run-time environment, comprising:
- determining whether a requested functionality is inherent in the class object;
  - receiving a request from an application for functionality that is not inherent in the class object;
  - determining if the functionality is available in a first extension object; and
  - directing the request to the functionality in a second extension object, when the functionality is not available in the first extension object.
35. (Previously presented) The method of claim 34, further comprising obtaining an extension package having computer-executable instructions associated with the extension object functionality.
36. (Previously presented) The method of claim 35, further comprising storing the extension package in an extension database.
37. (Previously presented) The method of claim 35, further comprising registering the extension package in an extension database.
38. (Previously presented) The method of claim 34, wherein the extension package proffers an extension provider object when the functionality is requested.

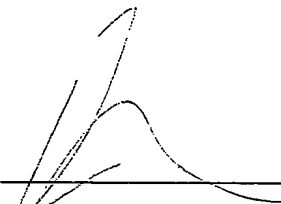


39. (Previously presented) The method of claim 38, wherein the extension provider object creates the extension object.
40. (Previously presented) The method of claim 34, further comprising searching for an entry associated with the functionality in an extension database.
41. (Previously presented) The method of claim 34, further comprising creating the second extension object when the extended functionality is first referenced, and locating the second extension object when the extended functionality is subsequently referenced.
42. (Presently Amended) A system for extending functionality of a class object, comprising:
- a processing unit;
  - a system memory in communication with the processing unit via a system bus;
  - a computer-readable medium in communication with the processing unit via the system bus;
  - an extensible object model executed from the computer-readable medium by the processing unit, wherein the extensible object model determines whether a requested functionality is inherent in the class object, and wherein the extensible object model creates an extension object from an existing extension package when a requested functionality is not inherent in the class object, and wherein the extension object extends the class object to provide the requested functionality.
43. (Previously presented) The system of claim 42, wherein information about the extension package is stored in an extension database.
44. (Previously presented) A method for extending functionality of a class object, comprising:
- determining whether a requested functionality is inherent in the class object;
  - invoking a functionality that is not inherent in the class object;
  - determining if the invoked functionality is available in a first extension object;

creating a second extension object when the invoked functionality is not available in the first extension object; and

directing the invocation to the second extension object.

Date: December 16, 2004



---

Vincent J. Roccia  
Attorney for Applicant  
Registration No. 43,887

Woodcock Washburn LLP  
One Liberty Place - 46th Floor  
Philadelphia PA 19103  
Telephone: (215) 568-3100  
Facsimile: (215) 568-3439